1. The program reads its own source file, which must be in the current directory.

```
import java.io.*;

class FileInputStreamDemo {

public static void main(String args[]) throws IOException {

int size;

InputStream f =new FileInputStream("FileInputStreamDemo.java");

System.out.println("Total Available Bytes: " + (size = f.available()));

int n = size/40;

System.out.println("First " + n +" bytes of the file one read() at a time");

for (int i=0; i < n; i++) {

System.out.print((char) f.read());

}

System.out.println("\nStill Available: " + f.available());

System.out.println("Reading the next " + n + " with one read(b[])");

byte b[] = new byte[n];

if (f.read(b) != n) {

System.err.println("couldn't read " + n + " bytes.");

}

System.out.println(new String(b, 0, n));

System.out.println("\nStill Available: " + (size = f.available()));

System.out.println("Skipping half of remaining bytes with skip()");

f.skip(size/2);

System.out.println("Still Available: " + f.available());

System.out.println("Reading " + n/2 + " into the end of array");

if (f.read(b, n/2, n/2) != n/2) {

System.err.println("couldn't read " + n/2 + " bytes.");

}

System.out.println(new String(b, 0, b.length));

System.out.println("\nStill Available: " + f.available());

f.close();

}
}
```

2. Demonstrate FileOutputStream

The following example creates a sample buffer of bytes by first making a **String** and then using the **getBytes( )** method to extract the byte array equivalent. It then creates three files. The first, **file1.txt**, will contain every other byte from the sample. The second, **file2.txt**, will contain the entire set of bytes. The third and last, **file3.txt**, will contain only the last quarter.

```
import java.io.*;
class FileOutputStreamDemo {
public static void main(String args[]) throws IOException {
String source = "Now is the time for all good men\n"
+ " to come to the aid of their country\n"
+ " and pay their due taxes.";
byte buf[] = source.getBytes();
OutputStream f0 = new FileOutputStream("file1.txt");
for (int i=0; i < buf.length; i += 2) {
f0.write(buf[i]);
}
f0.close();
OutputStream f1 = new FileOutputStream("file2.txt");
f1.write(buf);
f1.close();
OutputStream f2 = new FileOutputStream("file3.txt");
f2.write(buf,buf.length-buf.length/4,buf.length/4);
f2.close();
}
}
```

ByteArrayInputStream

3. The following example creates a pair of **ByteArrayInputStream**s, initializing them with

the byte representation of the alphabet:

```
// Demonstrate ByteArrayInputStream.
import java.io.*;
class ByteArrayInputStreamDemo {
```

```
public static void main(String args[]) throws IOException {
String tmp = "abcdefghijklmnopqrstuvwxyz";
byte b[] = tmp.getBytes();
ByteArrayInputStream input1 = new ByteArrayInputStream(b);
ByteArrayInputStream input2 = new ByteArrayInputStream(b,0,3);
}
}
```

The **input1** object contains the entire lowercase alphabet, while **input2** contains only the first three letters.

A **ByteArrayInputStream** implements both **mark( )** and **reset( )**. However, if **mark( )** has not been called, then **reset( )** sets the stream pointer to the start of the stream—which in this case is the start of the byte array passed to the constructor.

4. The next example shows how to use the **reset( )** method to read the same input twice. In this case, we read and print the letters "abc" once in lowercase and then again in uppercase.

```
import java.io.*;
class ByteArrayInputStreamReset {
public static void main(String args[]) throws IOException {
String tmp = "abc";
byte b[] = tmp.getBytes();
ByteArrayInputStream in = new ByteArrayInputStream(b);
for (int i=0; i<2; i++) {
int c;
while ((c = in.read()) != -1) {
if (i == 0) {
System.out.print((char) c);
} else {
System.out.print(Character.toUpperCase((char) c));
}
}
System.out.println();
in.reset();
}
}
}
```

ByteArrayOutputStream

ByteArrayOutputStream is an implementation of an output stream that uses a byte array as the destination. **ByteArrayOutputStream** has two constructors, shown here:

ByteArrayOutputStream( )

ByteArrayOutputStream(int *numBytes*)

In the first form, a buffer of 32 bytes is created. In the second, a buffer is created with a size equal to that specified by *numBytes.* The buffer is held in the protected **buf** field of **ByteArrayOutputStream**. The buffer size will be increased automatically, if needed. The number of bytes held by the buffer is contained in the protected **count** field of **ByteArrayOutputStream**.

6. The following example demonstrates **ByteArrayOutputStream**:

```
// Demonstrate ByteArrayOutputStream.
import java.io.*;
class ByteArrayOutputStreamDemo {
public static void main(String args[]) throws IOException {
ByteArrayOutputStream f = new ByteArrayOutputStream();
String s = "This should end up in the array";
byte buf[] = s.getBytes();
f.write(buf);
System.out.println("Buffer as a string");
System.out.println(f.toString());
System.out.println("Into array");
byte b[] = f.toByteArray();
for (int i=0; i<b.length; i++) {
System.out.print((char) b[i]);
}
System.out.println("\nTo an OutputStream()");
OutputStream f2 = new FileOutputStream("test.txt");
f.writeTo(f2);
f2.close();
System.out.println("Doing a reset");
f.reset();
for (int i=0; i<3; i++)
f.write('X');
System.out.println(f.toString());
}
}
```

BufferedInputStream

Buffering I/O is a very common performance optimization. Java's **BufferedInputStream** class allows you to "wrap" any **InputStream** into a buffered stream and achieve this performance improvement.

BufferedInputStream has two constructors:

BufferedInputStream(InputStream *inputStream*)

BufferedInputStream(InputStream *inputStream*, int *bufSize*)

The first form creates a buffered stream using a default buffer size. In the second, the size of the buffer is passed in *bufSize.* Use of sizes that are multiples of a memory page, a disk block, and so on, can have a significant positive impact on performance.

6. **Use buffered input**

The following example contrives a situation where we can use **mark( )** to remember where we are in an input stream and later use **reset( )** to get back there. This example is parsing a stream for the HTML entity reference for the copyright symbol. Such a reference begins with an ampersand (&) and ends with a semicolon (;) without any intervening whitespace. The sample input has two ampersands to show the case where the **reset( )** happens and where it does not.

```
import java.io.*;
class BufferedInputStreamDemo {
public static void main(String args[]) throws IOException {
String s = "This is a &copy; copyright symbol " +
"but this is &copy not.\n";
byte buf[] = s.getBytes();
ByteArrayInputStream in = new ByteArrayInputStream(buf);
BufferedInputStream f = new BufferedInputStream(in);
int c;
boolean marked = false;
while ((c = f.read()) != -1) {
switch(c) {
case '&':
if (!marked) {
f.mark(32);
marked = true;
} else {
marked = false;
}
break;
case ';':
if (marked) {
marked = false;
System.out.print("(c)");
} else
System.out.print((char) c);
break;
case ' ':
if (marked) {
marked = false;
f.reset();
System.out.print("&");
} else
System.out.print((char) c);
break;
default:
if (!marked)
System.out.print((char) c);
break;
}
}
}
}
```